NASA-CR-194818

November 1989

UILU-ENG-89-2239
CSG-115

COORDINATED SCIENCE LABORATORY
College of Engineering

PARAGRAPH: A GRAPHICS TOOL FOR PERFORMANCE AND RELIABILITY ANALYSIS

Kevin Douglas Lee

(NASA-CR-194818) PARAGRAPH: A GRAPHICS TOOL FOR PERFORMANCE AND RELIABILITY ANALYSIS M.S. Thesis (Illinois Univ.) 51 p

N94-71345

Unclas

Z9/61 0201486

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS None
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; distribution unlimited
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) UILU-ENG-89-2239 (CSG-115)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Coordinated Science Lab University of Illinois 6b. OFFICE SYMBOL (If applicable) N/A	7a. NAME OF MONITORING ORGANIZATION NASA
6c. ADDRESS (City, State, and ZIP Code) 1101 W. Springfield Avenue Urbana, IL 61801	7b. ADDRESS (City, State, and ZIP Code) NASA-AMES Research Center Moffett Field, CA 94035
8a. NAME OF FUNDING / SPONSORING ORGANIZATION NASA 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER NASA AMES NCA2-385
8c. ADDRESS (City, State, and ZIP Code) see 7b	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)

PARAGRAPH: A Graphics Tool for Performance and Reliability Analysis

12. PERSONAL AUTHOR(S)

Lee, Kevin Douglas

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Technical	FROM _____ TO _____	November, 1989	50

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	multi-level simulation, graphics, animation, interviews, X-windows

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

PARAGRAPH is an animated graphics display package. It consists of two parts: an interface to CSIM (a process based simulation language) and a graphic display system. When a simulation model is executed on CSIM the interface collects pertinent performance analysis data and writes them to a file. This file is then fed to the graphic display system which depicts the execution of the simulation model visually.

This report focuses on the graphical display system. Specifically it describes the user interface and features of the display system. It also explains how Interviews (which is based on X-windows) is used as the basis for the design of PARAGRAPH. The last section contains an example in which a basic load balancing simulation model is used to demonstrate the features and the capability of PARAGRAPH.

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (Include Area Code) / 22c. OFFICE SYMBOL

PARAGRAPH: A GRAPHICS TOOL FOR PERFORMANCE AND RELIABILITY ANALYSIS

BY

KEVIN DOUGLAS LEE

B.S., Purdue University, 1987

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 1989

Urbana, Illinois

"Funds for the support of this study have been allocated by the NASA-Ames Research Center, Moffett Field, California, under Interchange No. NCA2-385."

PAGE 11 INTENTIONALLY BLANK

ACKNOWLEDGEMENTS

I would like to thank Professor Iyer for his excellent guidance and encouragement. I would also like to thank my parents, Franklin and Rita Lee, and also Kumar Goswami for all of their help and support. A very special thanks goes to Anna Lam, my partner for life.

TABLE OF CONTENTS

CHAPTER **PAGE**

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Evaluating the performance of complex systems is a critical step in the design and planning of such systems. Design evaluation is often based on a performance model of the system. Several special purpose modeling languages have been developed to aid the performance evaluation process. With these languages, performance models can be quickly and easily constructed, evaluated and verified. Simulation environments are becoming more graphically oriented with the recent widespread availability of inexpensive graphic workstations. By displaying performance models pictorially, evaluation has been quickened and simplified.

In this thesis, a graphic viewing system that can be interfaced with a complex simulation environment to provide a real-time display of various system activities has been developed. Currently the system is linked to CSIM [1], a process oriented simulation language based on the C programming language. The interface to the simulation language is set up such that the graphics are not directly dependent on the simulation language.

The CSIM simulation language was slightly modified to interface with the graphics. To use this viewing package, a CSIM simulation is written with a few extra instructions for the initialization of the graphics. The CSIM simulation is executed, as usual, and two additional files are created, generating the graphics information. Using the information in these two files, the simulation can then be viewed in real time. In this context, real time refers to the simulation time. By displaying the simulation in real time, flaws and benefits of complex systems can often be seen that are not always noticed in the final statistics.

The current version of the graphic viewing system uses X11 windows running on a Sun workstation. The graphics are generated with the InterViews graphics libraries in C++.

This thesis is organized as follows. Previous work done with graphics oriented simulation environments will be discussed in Chapter 2. A high-level look at the design of this graphic viewing package, including the window and menu interfaces, will be discussed in Chapter 3. In Chapter 4, a more detailed look at the design of the viewing package will be discussed, including the data structures design. In Chapter 5, a load balancing algorithm simulation will be presented. Finally, some conclusions about this graphic viewing package will be made in Chapter 6.

CHAPTER 2

BACKGROUND

2.1 Previous Work

The need to simplify the design of complex simulation models has spurred research in graphically based simulation environments. Browne, Neuse, Dutton and Yu [2] designed a system which graphically simulates generalized queuing network models. This system allows users to graphically design complex systems, including computer and communications systems, by combining a specialized programming system and a graphical methodology. The Performance Analyst's Workbench System (PAWS) language eases the creation and evaluation of queueing network models. Information Processing Graphs (IPGs), a graphical methodology for modeling, is a directed graph where the nodes represent devices/resources and the arcs represent workloads flowing among the nodes. IPGs and the PAWS programming system were developed in conjunction with each other. A graphical editor is used to create the IPG by placing nodes, connecting the nodes with

arcs and assigning specifications for the nodes and arcs. From the IPG a PAWS program is generated and the model can then be executed.

Kurose and Gordon [3] designed a high-level interactive, graphics-oriented system based on the RESearch Queuing package (RESQ). The RESQ simulation language [4] is a special-purpose modeling language similar to PAWS. With this system, the model is also created with a graphics editor. Icons that have specific graphical-semantic definitions of some element of the simulation model are placed in the modeling area. Each instance of all icons has textual attributes assigned to it. The icons are connected together with lines representing the routing of jobs. A RESQ model is created from the graphical representation and is executed. The user can view graphically all of the performance measures of the simulation.

Melamed [5] created an interactive animated simulation package for queueing networks called the Performance Analysis Workstation (PAW). The PAW system simulates queueing networks with a discrete event simulation system. With a graphics editor, the user draws the nodes of the network and specifies the topology by connecting the nodes. The model is parameterized with a text editor by filling in captioned fields of forms. With forms, the user no longer needs to know any syntax. The actual simulation is executed by a Monte Carlo simulator. The simulation can be examined event by event. The PAW system animates the screen with the transfer of transactions between nodes and the creation and destruction of transactions. This feature is mainly used to verify

the validity of the model. When the model is determined to be valid, the animation is not required and only the final statistics are collected.

2.2 Graphic Viewing Package

The emphasis of the graphics in the systems discussed in the previous section has been on the input of the simulation model and the output of the simulation statistics. The analysis of the models mainly consists of the final statistics of the entire simulation which are displayed graphically. Unlike the systems described above, the graphics environment described in this thesis uses a separate simulator, for example, CSIM. The simulation details are sent to an output file which is used as input to the graphics environment to play back the progress of the simulation in real time. This is somewhat similar to the PAW system, but the system described in this thesis displays a variety of statistics in simulated real time as well as the transfer of transactions and communications between nodes. This allows the user to evaluate the performance at specific times of the simulation, not only at the end of the simulation. Several levels of detail can be viewed as in the PAW system, but the more detailed levels are not only used for validation of the model, but also in the evaluation of the model.

CHAPTER 3

USER INTERFACE

The graphic viewing package developed in this thesis can run with minimal intervention from the user, but an interface was built to allow the user to dynamically customize the simulator display to view specific areas of interest. The general flow of the viewing system is shown in Figure 3.1. All figures appear at the end of the chapter. First the initialization file is read and the simulation is initialized. Once the simulation starts to execute, the user has a choice of opening more detailed windows or altering the view of any window. Initially, only one window is open but up to four more can be opened. All commands are menu-driven via the mouse. If there is no user command pending, the simulation reads data from the log file and updates the graphics display.

3.1 Windows

A total of five different windows are in this graphic viewing system. Each window displays a different level of detail of the simulation. The window hierarchy is shown in

Figure 3.2. The available levels are the network level, the physical level, the selected physical level, the detailed level and the selected detailed level.

The network level is considered the main level because it is always displayed, and because of its versatility since almost all of the simulation data can be viewed from this level. It is the only window that is dependent on the architecture specified in the initialization file. Currently there are three different architectures supported: single-bus, a 16-node hypercube and no-connection. An example of a single-bus architecture is shown in Figure 3.3 and an example of a hypercube architecture is shown in Figure 3.4. The no-connection architecture is very similar to the single-bus architecture except no connections are shown between the nodes.

Each node can contain a variety of different facilities such as CPUs, disk drives, tape drives, etc. Each facility can have multiple processes running on it. Each process can have multiple mailboxes receiving communications from other processes. Facilities, processes and mailboxes are each represented separately as graphic objects. Within each node of the network level window, any of the facilities, processes or mailboxes contained in that node can be displayed. The object that is displayed can be selected from the node menu which will be discussed in more detail in Section 3.2.2.

All inter-node communications can be shown at the network level. In the initialization file, the user defines the message types that will be used in the simulation up to a maximum number of five. Each message type is assigned a color or pattern and is displayed on the left side of the network level window. In Figure 3.3, only three different

message types are defined by the user thus only the three different patterns are displayed with the corresponding message type names. In Figure 3.4, all five message types are defined.

The physical level and the selected physical level display all the facilities in the entire simulation and the facilities of selected nodes, respectively. For the selected physical level, the nodes are selected in the network level window by using the node menu. This is useful for viewing only one node or a few specific nodes. An example of a physical level window is shown in Figure 3.5.

The detailed level, as the name suggests, displays the most details of the simulation. All of the facilities, processes and mailboxes and all communications are displayed. Unlike the network level, this level can show all intra-node communications as well as all inter-node communications. The selected detailed level, like the selected physical level, displays only the facilities, processes and mailboxes of selected nodes. The communications that are displayed in the selected physical level can originate and terminate in nodes that are not selected and therefore not displayed. In this case the messages are shown heading towards the edge of the window. A history of the last four messages is shown in this level. The message types are also coded by the same color or pattern as in the top level. In Figure 3.6, an example of a detailed level window is shown. This example has a single CPU in each node. The last three messages are shown to be originating from node number one and are the same type.

All of the windows have a status box made entirely of text at the top of the windows. The status box displays the name of the window, the names of the initialization and log files, the simulation clock, the communication status, the pause status, and up to two possible simulation messages. The communication status refers to the display of communications in that window. If the communications are set to on, all of the messages are displayed, and conversely, if the communications are set to off, none of the messages are displayed. This feature applies only to the network, detailed and selected detailed levels since these are the only windows that display communications. In Figure 3.3, information in the status box can be interpreted as a network level window with lbal.init and lbal.log as the initialization and log files, respectively. The communications are currently on, the simulation clock is at 15.0 and the simulation message is *nd: 0 xfer job 182904*.

The amount of information to display can become larger than the screen size, especially in the detailed windows. For this reason, all the windows have vertical and horizontal scrollers along the right and bottom sides of the window. By holding down the middle button of the mouse on a scroller, the user can change the view of the window. The white box inside the gray box represents the current view relative to the entire virtual view. For example, in Figure 3.5, the white part of the horizontal scroller covers half of the entire scroller and is positioned against the left side. This means the current view is of the leftmost half of the entire virtual window and to the right, the undisplayed portion of the window is about the same size as the portion currently viewed. Similarly,

the vertical scroller shows that currently only about a fifth of the entire virtual window is being displayed. The windows are automatically sized according to the number of objects to be displayed. If there are too many objects in a window to be displayed on the screen at the same time, the window is sized so that the entire window is shown on the screen and the user may scroll around to see the rest of the window. All of the windows may also be resized using regular X-windows commands, but the image inside the window does not shrink or enlarge along with the window.

3.2 Menus

The menus are hierarchically organized in the sense that a submenu appears after selecting most of the top-level menu commands. Two different top level menus are available. The main menu can be selected from anywhere on the screen and provides a majority of the user interface. The node menu is available only in the network level window by clicking the mouse on a node. Each menu and its respective submenus are described in detail in the following sections.

3.2.1 Main menu

Most of the dynamic changes in the simulation are done with the main menu (shown in Figure 3.7). The main menu is obtained by pressing the right mouse button down anywhere on the screen and holding it down. Selections are made by *dragging the mouse* to the desired selection and releasing the button. The first selection in the main menu is *Pause*. This toggles the pause feature. Selecting this when the simulation is running will

cause the simulation to stop until this selection is made again. This is useful in closely examining the details of the simulation and reconfiguring the dynamically changeable parameters of the simulation.

The second choice, *Select Speed*, displays the dialog box shown in Figure 3.8. This dialog box is also displayed at the beginning of the simulation in order to pick the initial speed of the simulation. The simulation speed can be changed throughout the entire simulation. Different speeds are attained through nested delay loops. The fastest speed does not go through any delay loops.

In all the windows except the network level window, the facilities display the mean queue length, utilization, mean response time and mean service time graphically with bars. Since the values of the mean queue length, mean response time and mean service time can be anything from a very small to a very large value, the ranges for each of these metrics can be specified dynamically by the user. The ranges can be changed through the main menu selection, *Select Ranges*. There is no need to change the range for the utilization because the value will always be between 0 and 1. A submenu is displayed (Figure 3.9), allowing the user to choose the metric whose range is going to be changed. A dialog box is displayed for both the high and low values. The dialog box for changing the high mean queue length value is shown in Figure 3.10. For all of these dialog boxes, the cursor needs to be in the dialog box in order for the keyboard input to be read. If the number is highlighted, anything typed on the keyboard will replace the old number.

To highlight the number, just drag the mouse over the number using any button. After changing the ranges, all the facilities will be updated with the new ranges.

Any or all of the windows can be redisplayed with the *Redisplay Windows*. The submenu shown in Figure 3.11 allows the user to choose a single window or all of them to be redisplayed. Since the network level window is the only window that is always displayed, a user can pick a window that is not currently open. Nothing is redisplayed when an unopen window is selected.

To turn off the communication messages, select the main menu entry *Comm. On/Off*. The submenu shown in Figure 3.12 will be displayed. Each window that has communications can have its messages turned off individually. This allows the user to see communications in one window and not in another if desired. Turning off the communications can greatly speed up the simulation. Again, a window can be chosen that is not currently open. In this case, turning off the communications will have no effect.

To open or close any window except the network level window, select *Open/Close Windows* in the main menu. A submenu will appear (Figure 3.13), listing the four different windows available. Selecting a window will toggle that window's status. For example, if the physical level window was not open, selecting the physical window would open it. Otherwise, if it was already open, it would close. The network level window is always displayed; therefore it cannot be opened or closed.

The method for exiting the graphic viewing package is provided in the main menu. Select *Exit* and all of the open windows will be closed.

3.2.2 Node menu

The node menu operates on only one node at a time. For example, if there are multiple nodes in the network level window, and the cursor is moved to a particular node, pushing the left mouse button displays the node menu. All of the functions of the node menu would then operate only on the node that was clicked upon. The node menu (Figure 3.14) serves two purposes. First, the user can select the facility, process or mailbox, which is contained in that particular node and which is to be displayed in the network level node. Second, the user can select nodes for inclusion in either the selected physical level window or the selected detailed level window.

Selecting *Facilities*,*Processes*, or *Mboxes*, causes a submenu listing the names of all the facilities, processes or mailboxes contained in that node to be displayed. Each node will have unique submenus. Selecting one object will cause that object to be displayed in the node at the network level. An example of a facilities submenu is shown in Figure 3.15.

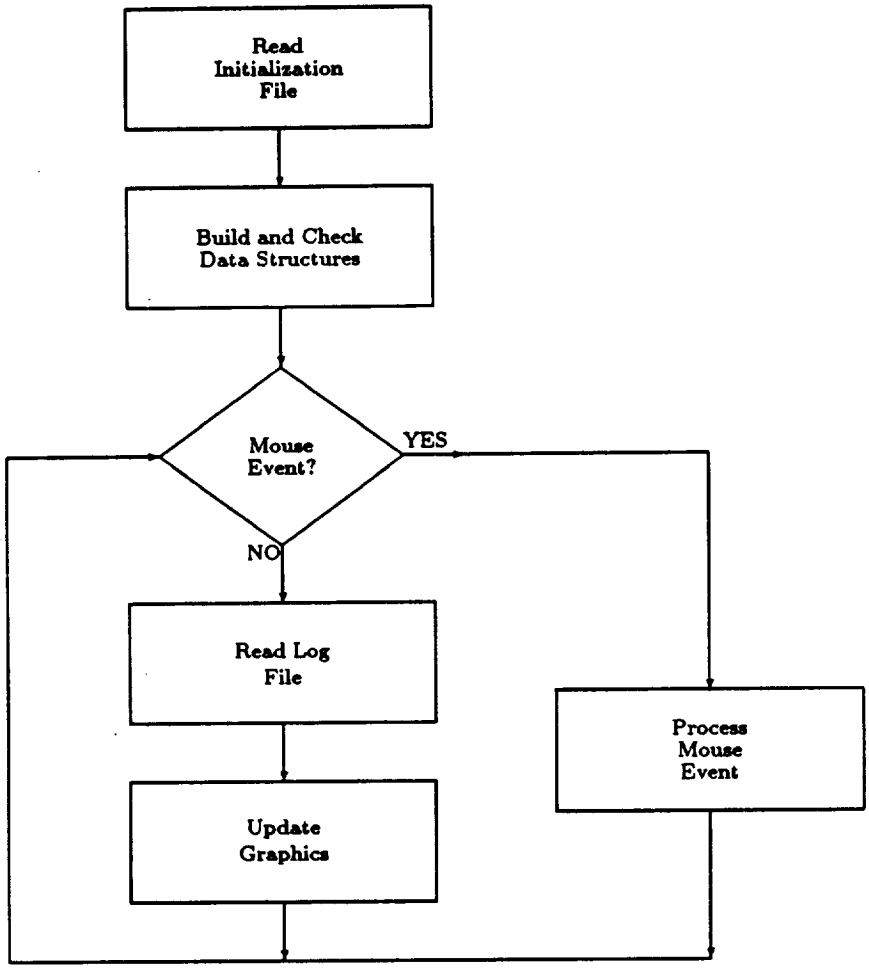

Figure 3.1: Program Flow Chart

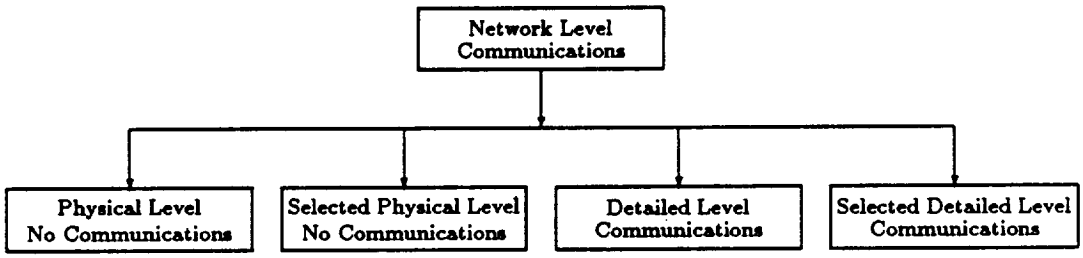

Figure 3.2: Window Hierarchy

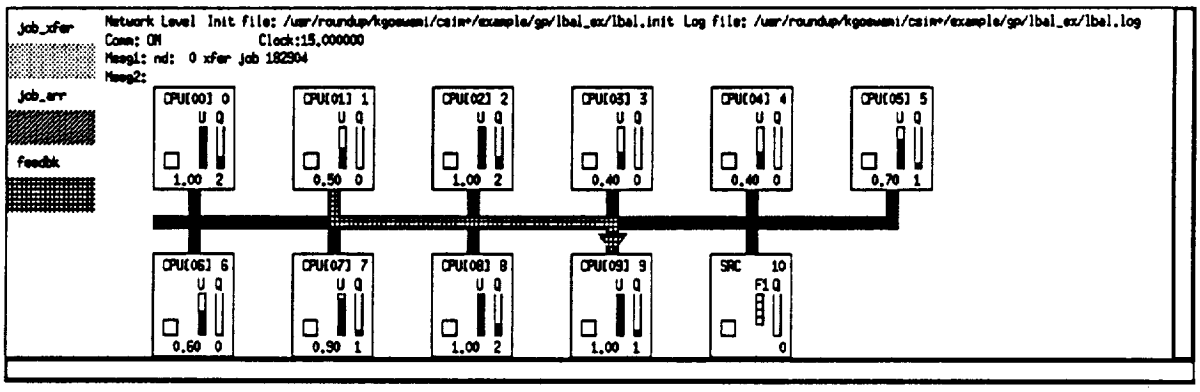

Figure 3.3: An Example of a Single Bus Network Level Window

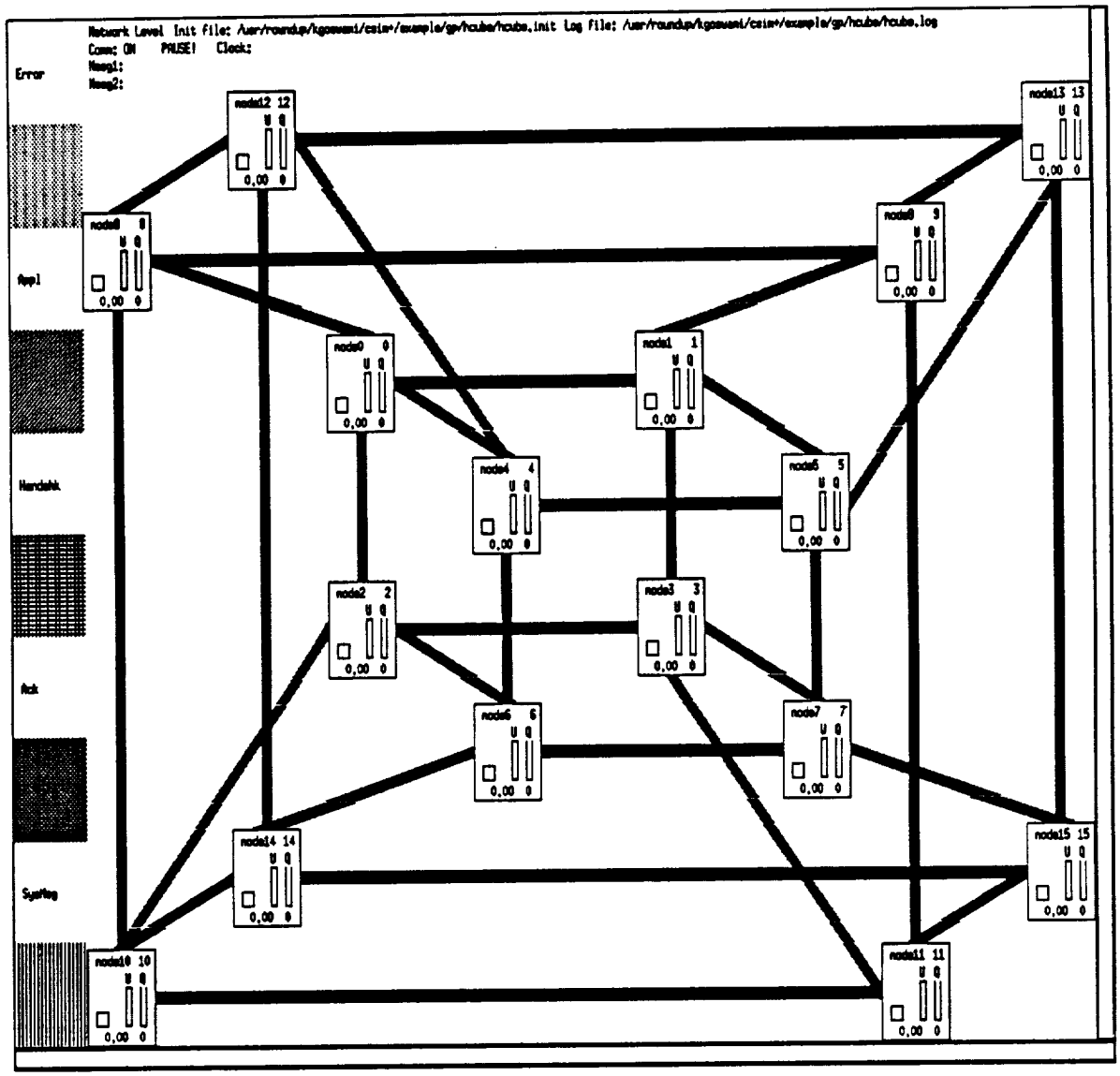

Figure 3.4: An Example of a Hypercube Network Level Window

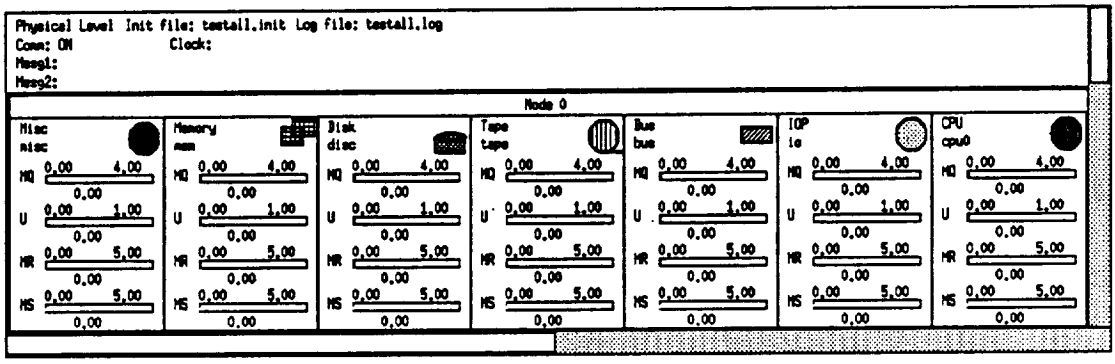

Figure 3.5: An Example of a Physical Level Window

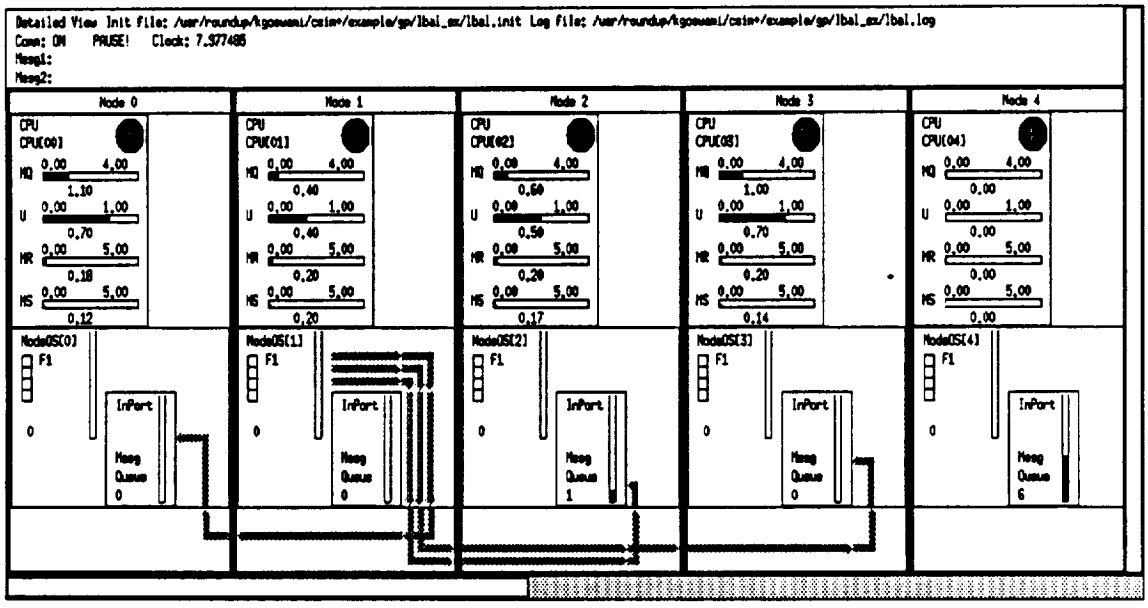

Figure 3.6: An Example of a Detailed Level Window

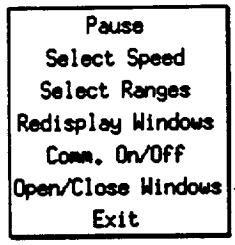

Figure 3.7: Main Menu

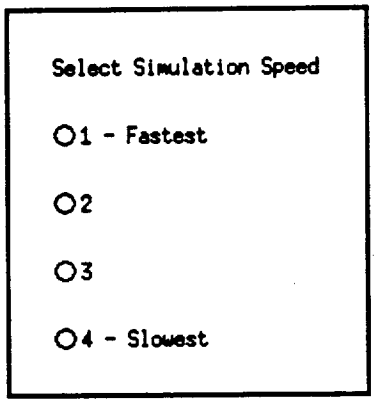

Figure 3.8: Speed Selection Dialog Box

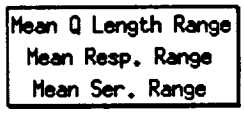

Figure 3.9: Select Ranges Submenu

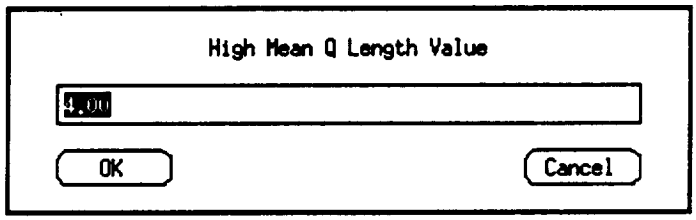

Figure 3.10: An Example of a Range Selection Dialog Box

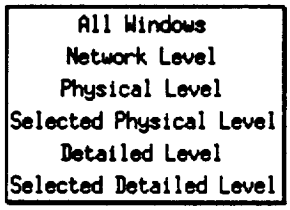

Figure 3.11: Redisplay Window Submenu

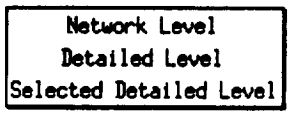

Figure 3.12: Communications On/Off Submenu

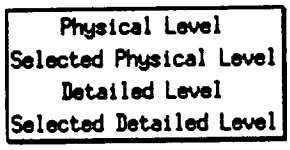

Figure 3.13: Open/Close Windows Submenu

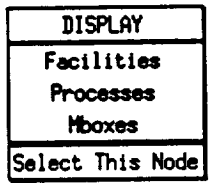

Figure 3.14: Smallnode Menu

Figure 3.15: Smallnode Facilities Display Submenu

CHAPTER 4

DESIGN

The development of this graphic viewing package is based on InterViews [6], a graphical user interface toolkit. InterViews is a library of C++ [7] classes that define common interactive objects and composition strategies. The InterViews classes are based on the X windowing system, but the developer is totally removed from any interaction with the X windowing system. The application's user interface is defined in terms of InterViews objects, which communicate with the windowing and operating systems.

InterViews provides three different types of objects: interactive type objects such as menus and buttons, structured graphic type objects such as circles and polygons, and structured text objects. New classes were created for this simulator by deriving them from the base classes provided by InterViews. The new classes are discussed in detail in the following section.

4.1 Classes

A majority of the derived classes are the interactive type, derived from the InterViews class Scene. This base class provides member functions called Redraw, Resize, Reconfig and Handle that are used in the derived classes. Redraw is used to draw all of the graphics in that object by calling InterViews routines to draw lines, rectangles and other geometric objects at specific locations. Resize recalculates the locations of the graphics in the object if the object size happens to be changed. Reconfig determines the object's size, stretchability and shrinkability. Handle takes care of all the key and mouse events. All of the graphic classes derived from Scene use derived versions of these routines. One advantage of a Scene derived class is the ability to insert other graphics objects into it. For example, two Scene derived classes in InterViews are HBox and VBox. When multiple objects are inserted into an HBox, they form a horizontal row of objects. The VBox class creates vertical rows of graphic objects. Together, the HBox class and the VBox class can be used to construct complex graphic displays. This is how the windows are created.

The main class is called Idraw[1]. As far as the graphics and the user interface is concerned, Idraw creates the Network level window and handles the main menu mouse inputs. Idraw also takes care of some non-graphic tasks, such as reading in the initial simulation structure, building the data structures and checking the correctness of the simulation structure.

[1]The name Idraw came from a program written in InterViews on which the structure of this simulator was originally based. Now the two Idraw classes barely resemble each other.

A new class was derived for each of the additional windows. These classes are constructed when the user requests a new window to be added through a mouse operated menu.

Several classes were derived which represent different types of CSIM objects. Each of these classes has a special member function called Updatedata. This routine updates all of the data in the object without redrawing the entire object. In other words, only the data that are being updated are redrawn. This is to speed up the updates and reduce the flashing effect on the screen.

The Facility class is an interactive graphics object that displays information on CSIM facilities. The Facility class was subclassed into specific types of facilities such as CPU, disk, tape, memory, etc. An example of a CPU facility is shown in Figure 4.1. Facilities display mean queue length, utilization, mean response time, mean service time, and also a little icon depending on the specific facility type. All of the different available facility types can be seen in Figure 3.5.

The Process class represents a process that runs on a facility in the same way as a CSIM process. An example is shown in Figure 4.2. Processes belong to a certain facility and display four flags and a queue. The four flags are independent binary flags that the user can set or reset individually or as a group. The meaning of these flags is determined by the user. Processes generate all of the communication messages in the simulation.

The last class that is directly related to a CSIM object is Mbox. Figure 4.3 shows an example of a Mbox. Mboxes represent mail boxes that receive communication messages. They belong to a Process and display a queue which represents the communication messages that have been received and are waiting for processing.

The network view window has a special graphics class that represents each node called Smallnode. Each Smallnode can display any facility, process or mbox that is contained in that node. An example of a Smallnode displaying a facility called *cpu[00]* is shown in Figure 4.4. The user can dynamically choose which object to display through a mouse-driven menu.

All of the windows which display communication messages have special communication classes. In the network level, these classes display the connectivity of the nodes along with the transfer of messages. The messages are either color coded on color systems or pattern coded on black and white systems for the different types of messages. The methods used for displaying communication messages are discussed in more detail later in this thesis.

4.2 Data Structures

The data structures are designed to allow fast updates of the graphics and also retain the structural dependence. For fast graphic updates, pointers are needed for each graphic object. An easy method for obtaining the correct pointer for each object is also needed. This is accomplished by using hash tables for the facilities, processes and mailboxes. The

objects are hashed on their ID number and the hashing algorithm is a modulo arithmetic approach. The ID number modulo the size of the hash table results in the hash table entry number. Each hash table entry is a linked list of graphic objects. The next pointer of the linked list is built into each graphic class. Therefore, in the case of collisions, a linked list is traversed to find the correct graphic object. A graphic object is created for each facility, process and mailbox despite the fact that at the onset of the simulation they are not displayed. These original graphic objects are used in the detailed window and they always exist. They contain all of the up-to-date information, e.g., facility utilization and mean queue length, process flags, mailbox queues, etc. When the detailed window is created, these original objects are used and inserted into the window. For all the other new windows, new auxiliary graphic objects are created, the vital information is copied into the new objects, and they are then inserted into the new window. Pointers to the new objects are kept in the original graphics object. These auxiliary objects are deleted when the window is closed. When updating the data, the original objects are updated first, then, if any other windows are open, the auxiliary objects are updated.

The smallnodes used in the network level are linked directly to the original graphic objects by pointers. The original graphic objects provide the simulation data needed to be displayed by the smallnode. This is how the different objects can be displayed in the same node. When a different facility, process or mailbox, is chosen to be displayed, a special pointer is set to the correct graphic object so that the smallnode can obtain the data needed for display. Therefore, the smallnode actually contains no data in itself.

It gets everything from the original objects. Thus, to update the simulation data, the original objects are updated first and then the smallnodes are redrawn. The new data will be used when the smallnode is redrawn.

The original graphic objects also contain the structural dependence information. Each node has a linked list of facilities that are contained within that node, each facility has a linked list of processes that are contained in that facility and lastly, each process has a linked list of mailboxes in that process. Also, in the graphic objects are the ID numbers of its structural parent. For example, each process has its facility's ID number and its node number.

The original graphic objects are created from the initialization file. The structural dependence is checked to see if there are any contradictions. For example, a mailbox may belong to a certain process in node A, but the initialization file may say the mailbox belongs to node B. If there is a discrepancy, a warning is printed and the data structure is modified to a point were the contradictions are eliminated.

4.3 Communication Messages

Communication messages are displayed at two different levels: the detailed level and the network level. The network level communications are dependent upon the architecture type while the detailed level communications remain the same regardless of the architecture.

4.3.1 Detailed level

In the detailed level, the messages always originate at a process and arrive at a mailbox. There are three different types of graphic objects used to display a message. First, there is the message origination point called a *Port*. The remaining two graphic objects are vertical and horizontal communication objects called *Vcomm* and *Hcomm*, respectively. The vertical communication objects run vertically next to the mailboxes, processes and ports. The horizontal communication objects are in a single row at the bottom of the window, connecting all of the different facilities. An example of the placement of the graphic objects can be seen in Figure 4.5.

In order to quickly draw messages, a method was needed to obtain pointers to the necessary communication objects. Thus, each process object has a pointer to a port object and each port has a pointer to a Vcomm object. A column of Vcomm objects is actually a linked list. Therefore, when a message is going to a different facility, the column is traversed until the end, drawing the message. Each Vcomm object may also be addressed by its adjacent mailbox. In other words, every mailbox has a pointer to the Vcomm object directly to the right of it. This is to facilitate the termination of messages.

The Hcomm objects are used only when a message is sent to a different facility. An array of Hcomm objects is created after the total number of facilities is first known. The size of the array is four times the number of facilities in the simulation. Half are used for the detailed window and the other half are used for the selected detailed window. For every facility, an Hcomm object is used under the processes and another is used under

the mailboxes. By knowing the column of the originating facility and the column of the terminating facility, the message can be drawn on the proper Hcomm objects.

4.3.2 Network level

In the network level, all the communications are dependent upon the architecture type. The single bus and the unconnected architecture types are considered the same as far as communications are concerned, meaning the messages are drawn the same. The only difference is whether or not the bus is drawn. Therefore, presently, only two different types of communications exist in the network level: single bus and hypercube.

The single bus messages are relatively simple. There are basically two different types of communication objects, a vertical one and a horizontal one called Vtopcom and Htopcom, respectively. A Htopcom separates two rows of nodes. There are always an even number of rows except when there are less than four nodes. In this case, there is only one row of nodes. An example of a sixteen-node layout is shown in Figure 4.6. The Htopcom knows how many nodes are connected to its top side and to its bottom side. The Vtopcom objects connect multiple Htopcoms. By knowing the columns of the originating and terminating nodes, and the originating and terminating Htopcom objects, a message can easily be drawn.

The hypercube network is more complex and is limited to the sixteen-node case. Only one type of communications graphic object is used: Tophcom. The different Tophcom objects are differentiated by their ID numbers. The placement of the Tophcom objects and the smallnodes are shown in Figure 4.7. Using the ID numbers, the size and location

of the Tophcom objects is known. This information is used to draw the connections and the messages. Since messages may have to traverse many different graphic objects, a method was obtained to identify paths from node to node. In each smallnode object, two arrays of sixteen pointers to Tophcom objects exist. One array points to the first Tophcom object in the path to a certain node and the other points to the last, called startcom and endcom, respectively. For example, for the path from node 4 to node 12, the twelfth entry in the startcom array in smallnode 4 would contain the pointer to the first Tophcom object (Tophcom 9) in that path. The array entries that correspond to the nodes that are not directly connected to this node are set to nil. The Tophcom objects are connected in linked lists depending on which path is desired. There exist thirty-two unique paths from node to node. The path traversal always originates from a lower numbered node and terminates at the higher numbered node. In order to draw a message, only a linked list needs to be traversed. A Tophcom object knows from its ID number and the node-to-node path where to draw a message.

4.4 Faults

The simulator can display two different types of faults. The first is a facility fault and the second is a node fault. Both fault types are displayed in the network level as an *"X"* across the entire node. The two types of faults are differentiated by either color or pattern. A node fault is either red or solid black, while a facility is either green or gray.

In the physical and detailed level windows, the facility fault is also displayed. It is also drawn as an *"X"* across the entire facility in either green or gray.

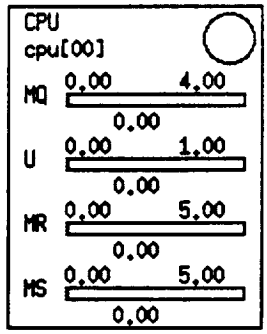

Figure 4.1: An Example of a CPU Facility

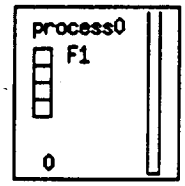

Figure 4.2: An Example of a Process

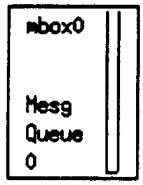

Figure 4.3: An Example of an Mbox

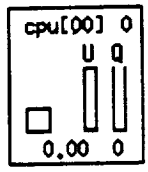

Figure 4.4: An Example of a Smallnode

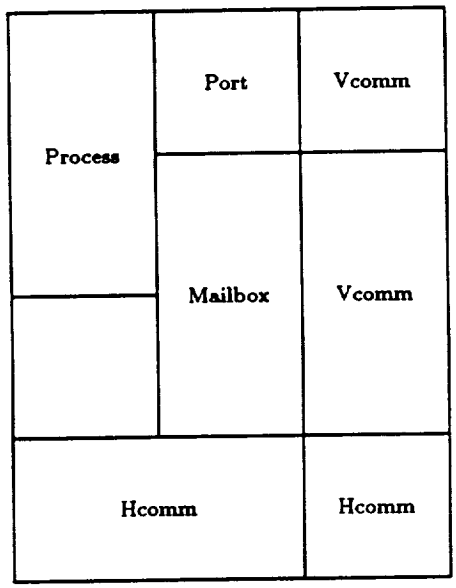

Figure 4.5: An Example of Detailed Level Communication Graphic Object Placement

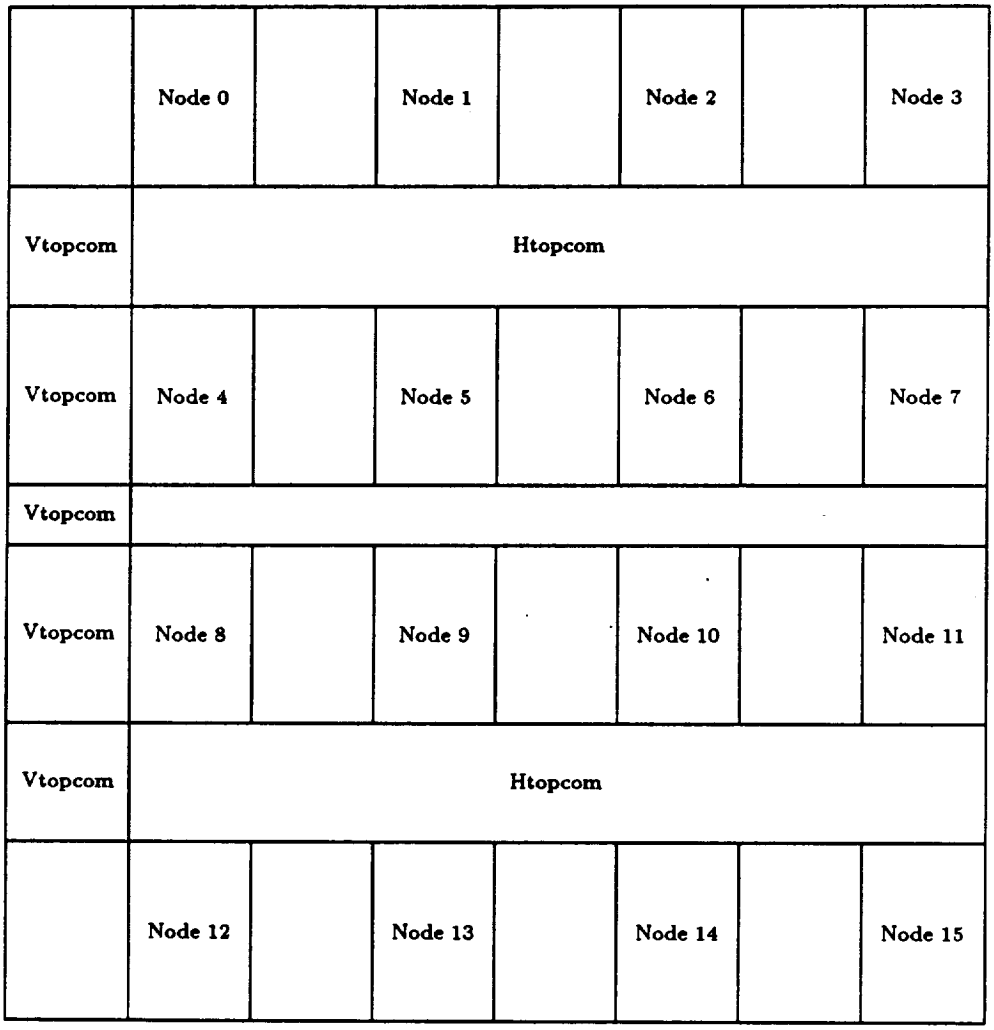

Figure 4.6: Network Level Single Bus Communication Graphic Object Placement

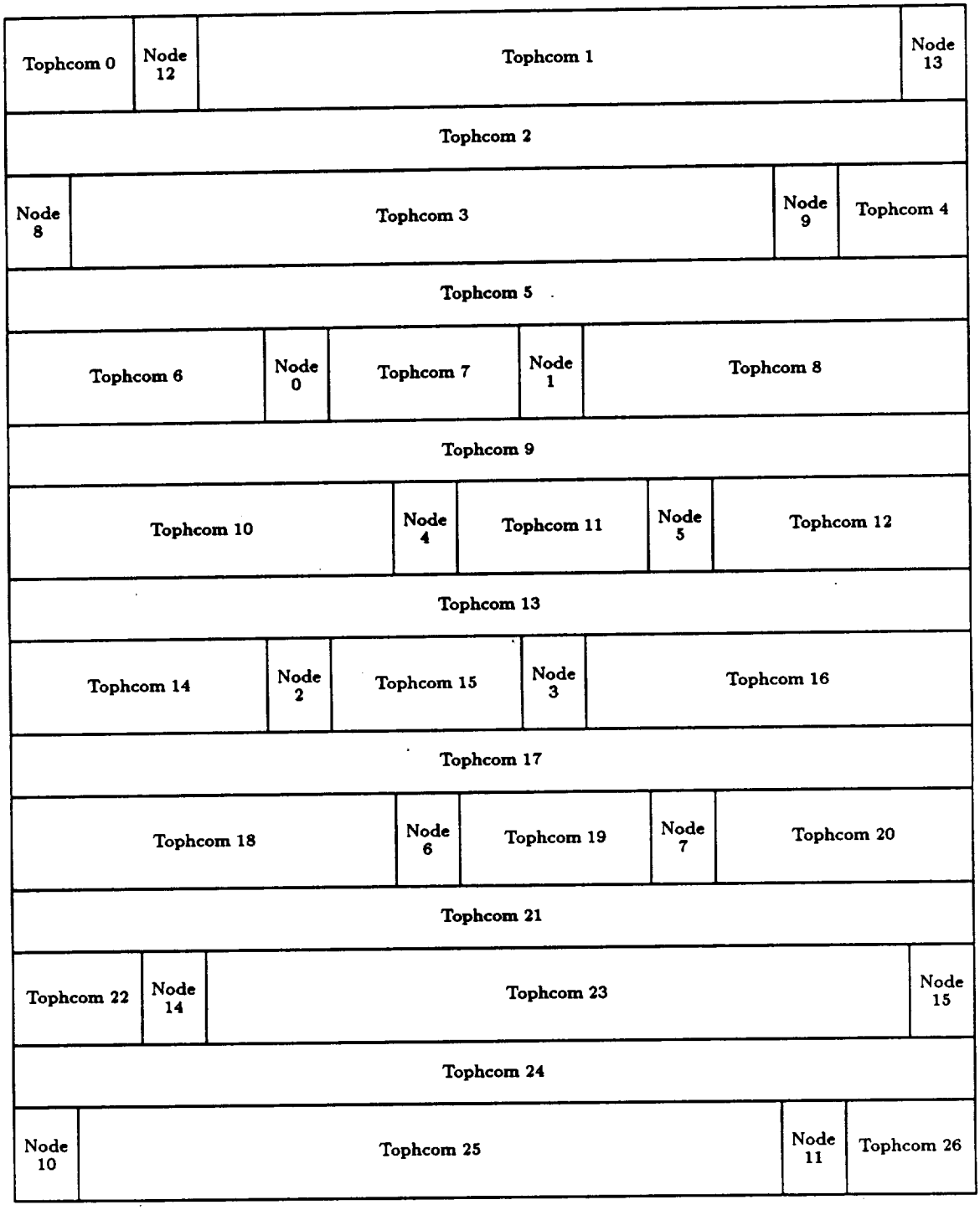

Figure 4.7: Network Level Hypercube Communication Graphic Object Placement

CHAPTER 5

A LOAD BALANCING SIMULATION

Load balancing is a method of spreading out the load among many different nodes in a distributed system. The main objective is to reduce the response time of the overall system by moving jobs from heavily loaded nodes to less heavily loaded nodes. Of course there are tradeoffs involved, such as the communication costs of sending system status information to all the nodes and transferring jobs from node to node. Many different load balancing algorithms exist but a simple one was chosen as an example of this graphic viewing package's capabilities.

The load balancing algorithm in this example runs on each individual node. Each node decides, when a new job is created at that node, whether or not to move the job to another node or process the job at that node. The algorithm makes its decision based on what it thinks is the size of each node's present mean queue length. Periodically, a node will broadcast its own mean queue length to all the other nodes. The tradeoff with this algorithm is the cost of frequent broadcasts versus up-to-date information about each

node. With more up-to-date information about each node a better decision can be made as to whether or not to transfer a job, and if the job is transferred, which node should it be transferred to. More details about this load balancing algorithm can be found in [8].

This example is made up of ten regular nodes and one general node. The general node (Node 10) is a special node for this simulation that does not exist in reality. It represents the work load generator for all the other nodes. Jobs are created in the general node and are sent to a node determined by a uniformly distributed random process. The work done by the general node uses no resources and does not affect the simulation. From a regular node's point of view, jobs are being created inside of itself. In Figure 5.1, a job is being created in the general node and sent to Node 9. When a node broadcasts its mean queue length to all the other nodes, it is displayed as many different messages, all of them originating from one node. One broadcast message, originating from Node 7, is shown in Figure 5.2. Figure 5.3 shows a job being transferred from Node 8 to Node 0. The algorithm is working correctly because Node 8 has a mean queue length of 3 and Node 0 has a mean queue length of 0. On closer inspection, it can be seen that Node 9 has a mean queue length of 0 and a utilization value of 0.0 while Node 0 has a utilization value of 0.4, indicating that Node 9 would probably be a better choice than Node 0.

As the simulation progresses, it is quite obvious that this load balancing algorithm is not very effective. Figure 5.4 shows Node 6, Node 8 and Node 9 with mean queue lengths of almost 2 and a utilization values of almost 1.0, while Node 3, Node 4 and Node 5 have mean queue lengths of 0 and utilization values of 0.0.

By displaying this simulation graphically, it was simple to verify that the algorithm was working correctly. It was also quite obvious that the algorithm was too simplistic to provide effective load balancing.

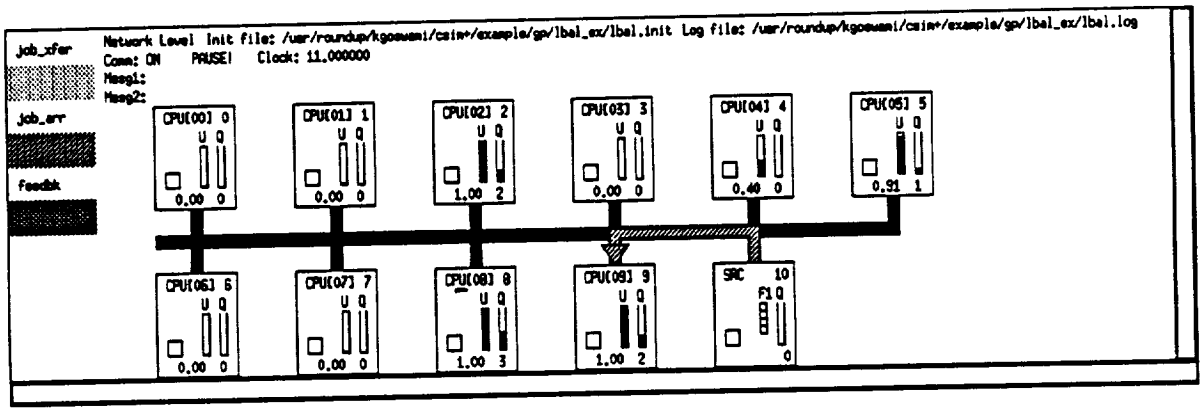

Figure 5.1: Arrival of a New Job

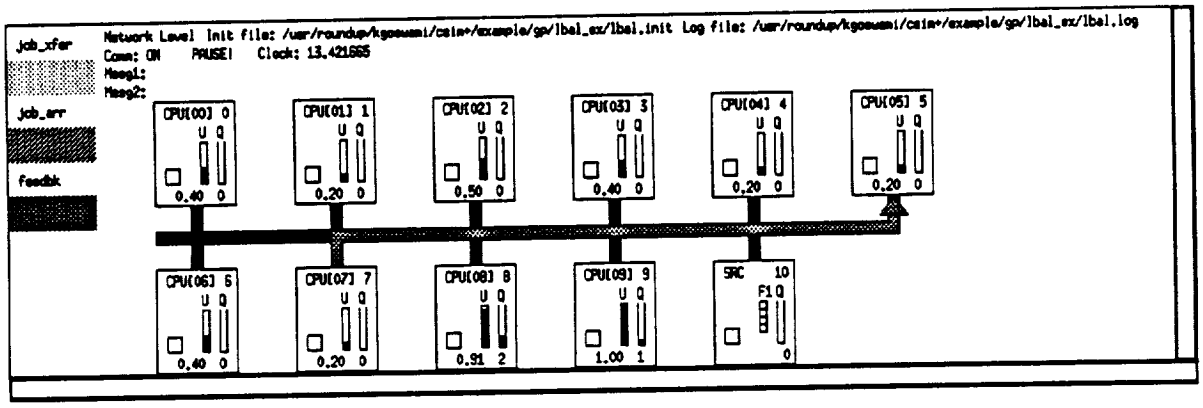

Figure 5.2: Broadcasting a Mean Queue Length

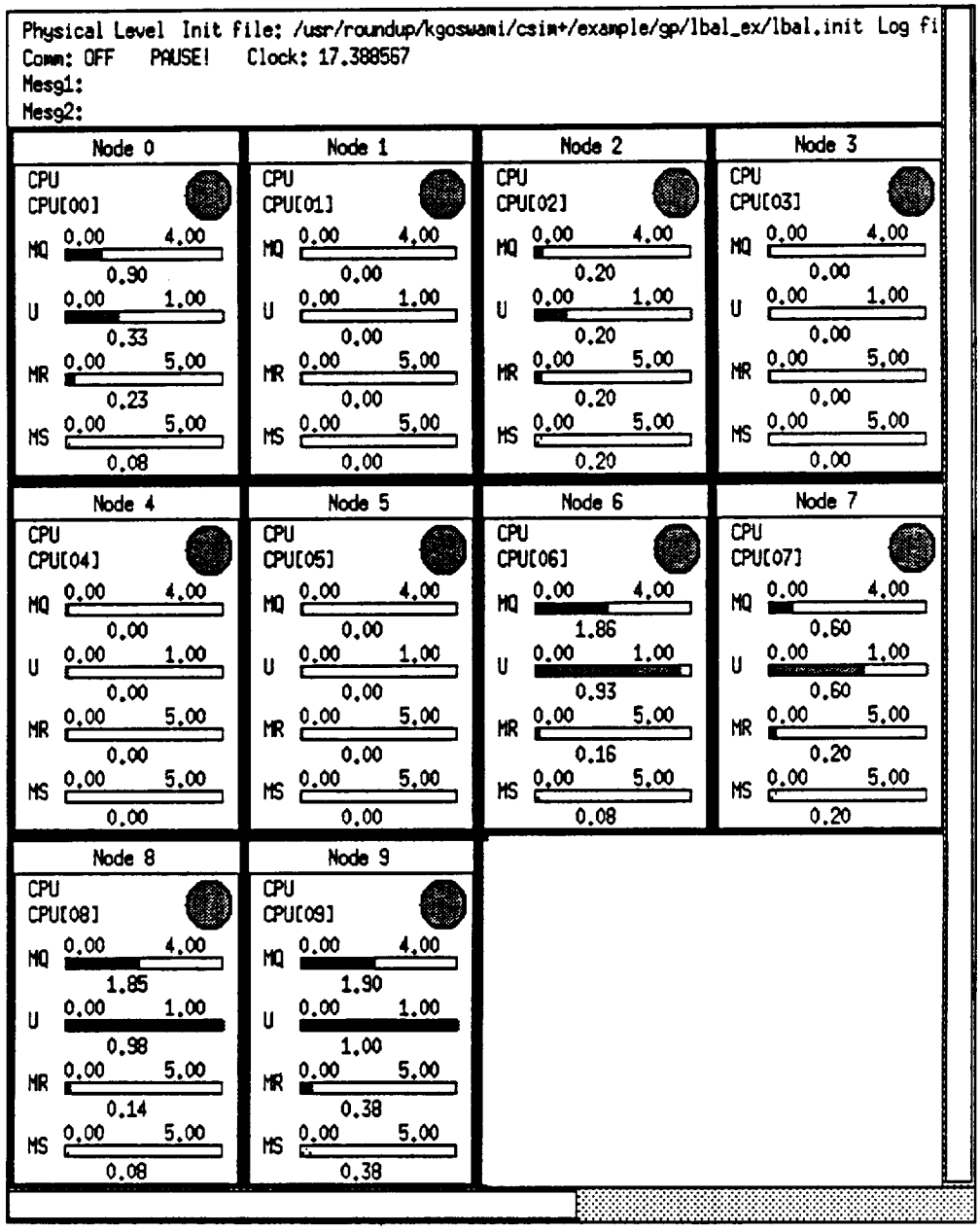

Figure 5.4: Load Balancing Physical Level

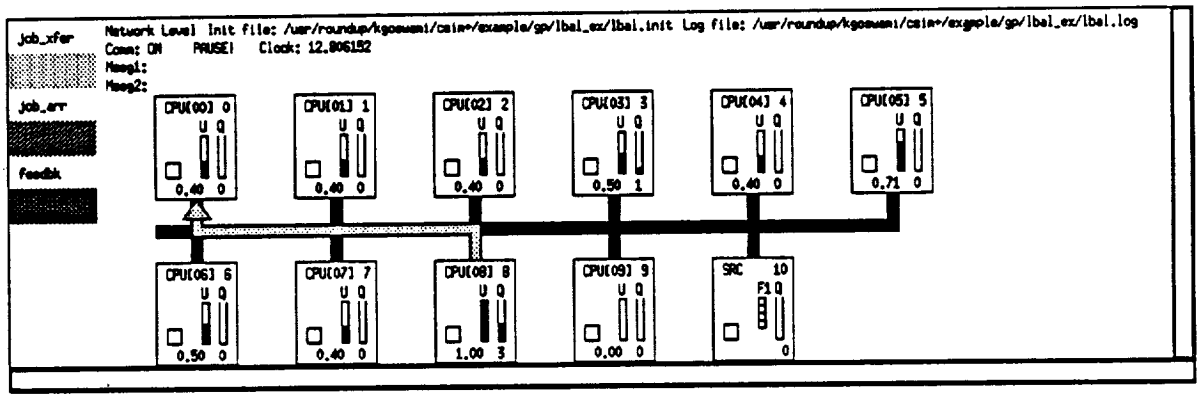

Figure 5.3: Transfer of a Job

CHAPTER 6

CONCLUSIONS

In this thesis, a graphic viewing system that can be interfaced to a complex simulation environment is described. The viewing system is currently linked to the CSIM simulation language. After developing a CSIM simulation, the simulation can be played back to show the progress of the simulation in real time. Analyzing the simulation in real time can give a more realistic view of the true behavior of the system. Some of the simulation features that can be viewed are the communication messages, the mean queue length, utilization, mean response time and mean service time of the resources, and message queues. Presently three different architecture types are supported: single bus, hypercube and unconnected.

In the future, this graphic viewing package could be expanded to handle more architecture types. Also, additional functionality could be added to the simulator's clock feature, e.g., allowing breakpoints to be set, and allowing the simulation time to be moved forward or backward.

REFERENCES

[1] H. Schwetman, "CSIM: A C-based, process-oriented simulation language," in *Winter Simulation Conference*, IEEE, 1986.

[2] J. C. Browne, D. Neuse, J. Dutton, and K. C. Yu, "Graphical programming for simulation of computer systems," in *Simulation Symposium*, IEEE, 1985.

[3] J. F. Kurose and K. J. Gordon, "A graphics-oriented modeler's workstation environment for the research queueing package (RESQ)," in *Fall Joint Computer Conference*, IEEE, 1986.

[4] C. Sauer, E. MacNair, and J. Kurose, "RESQ: CMS user's guide," Research Report RA-139, IBM, Yorktown Heights, N.Y., April 1982.

[5] B. Melamed, "The performance analysis workstation: An interactive animated simulation package for queueing networks," in *Fall Joint Computer Conference*, IEEE, 1986.

[6] M. A. Linton, J. M. Vlissides, and P. R. Calder, "Composing user interfaces with interviews," *IEEE Computer*, vol. 22, pp. 8–22, February 1989.

[7] B. Stroustrup, *The C++ Programming Language.* Reading, Massachusetts: Addison-Wesley, 1986.

[8] K. Goswami, "Load sharing base on task resource prediction." M.S. thesis, University of Illinois, Urbana, IL, 1988.